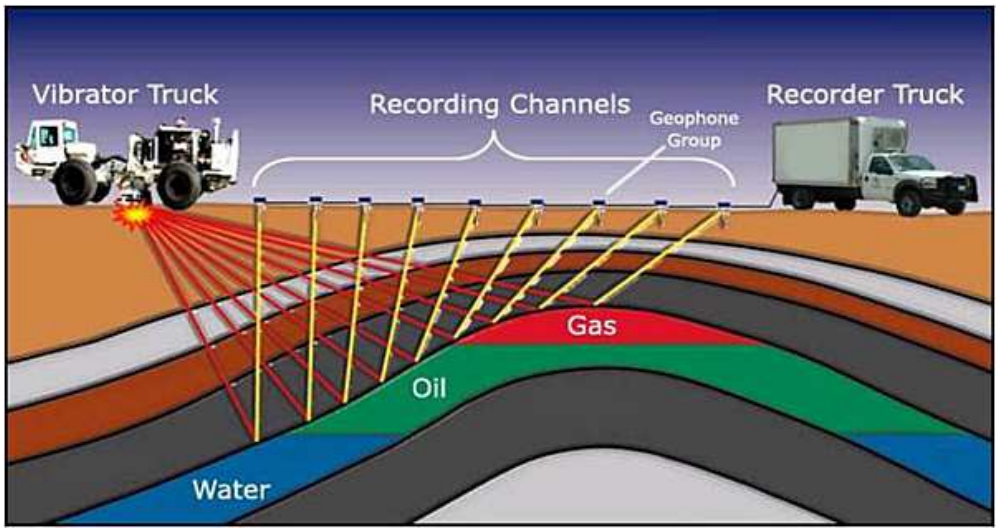
Magique-3D

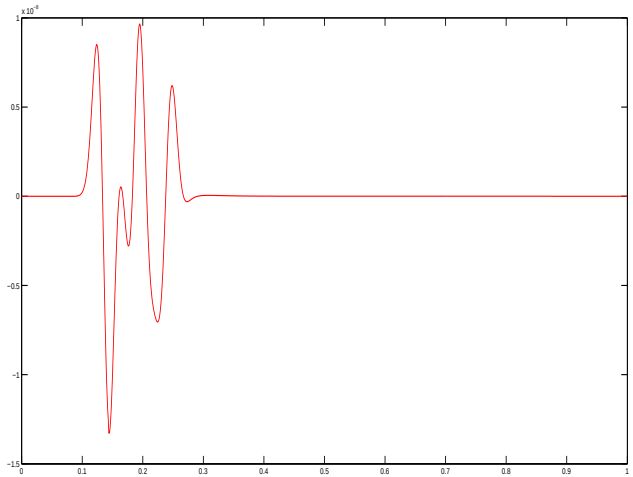
Imaging of complex media with elastic wave equations

H. Barucq, H. Calandra, J. Diaz, and **J.Luquel**

Context of the study : Seismic acquisition

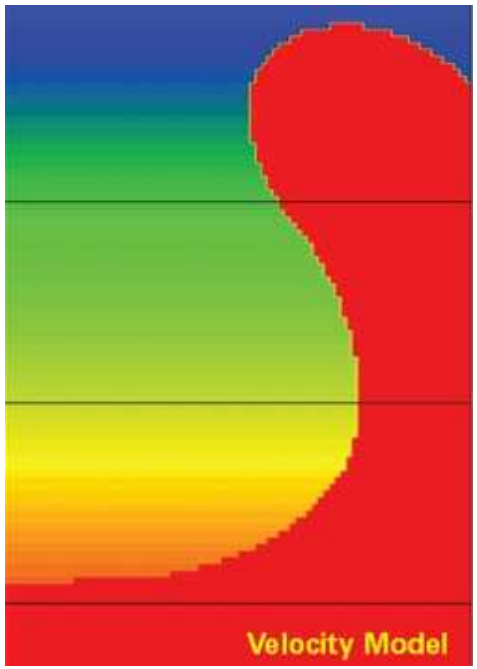


Acquisition

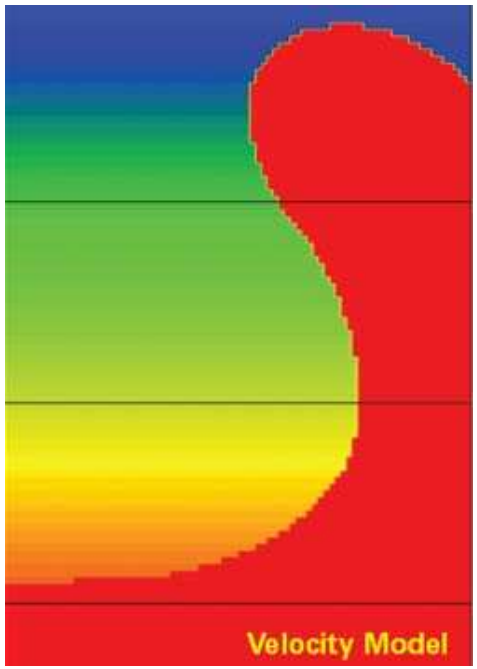


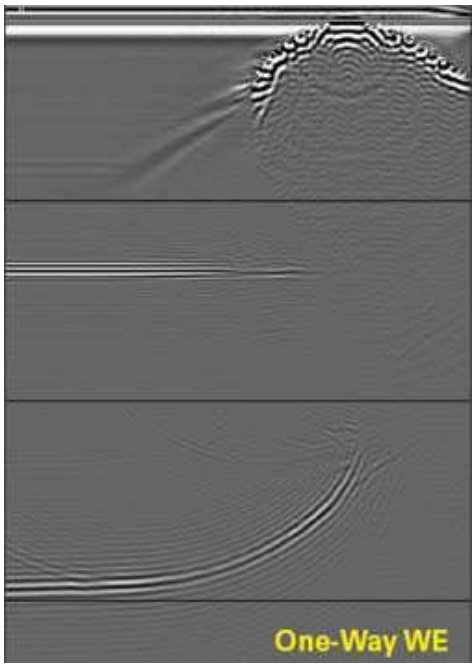
Seismogram

Numerical Methods of Wave Propagation



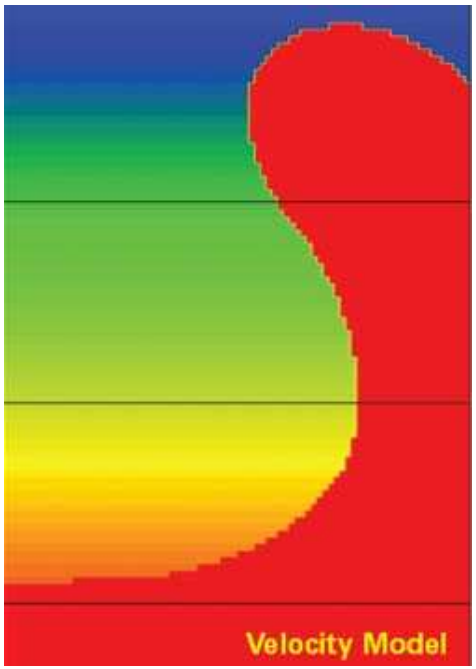
Numerical Methods of Wave Propagation

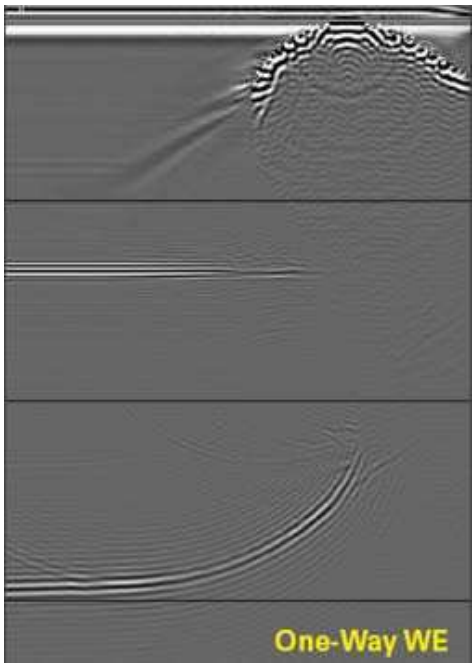


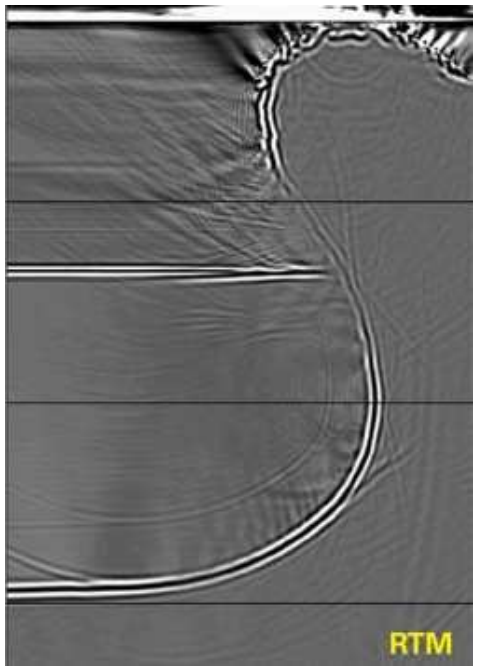


1. One-way

Numerical Methods of Wave Propagation







1. One-way
2. **Full Wave** (**R**everse **T**ime **M**igration)

Reverse Time Migration

- Technique for producing image of the subsurface
- Based on the solution of wave equations

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

2. By using sources and receivers of the seismic acquisition.

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

2. By using sources and receivers of the seismic acquisition.

   (a) **Propagate** sources into the velocity model :$\mathbf{u}_s^i(\mathbf{x}, t)$

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

2. By using sources and receivers of the seismic acquisition.

   (a) **Propagate** sources into the velocity model :$\mathbf{u}_s^i(\mathbf{x}, t)$

   (b) **Back-propagate** the wavefield recorded at receivers :$\mathbf{u}_r^i(\mathbf{x}, T - t)$($T$ is the final time)

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

2. By using sources and receivers of the seismic acquisition.

   We solve the equation **twice** for each source

   (a) **Propagate** sources into the velocity model :$\mathbf{u}_s^i(\mathbf{x}, t)$

   (b) **Back-propagate** the wavefield recorded at receivers :$\mathbf{u}_r^i(\mathbf{x}, T - t)$($T$ is the final time)

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

2. By using sources and receivers of the seismic acquisition.

   We solve the equation **twice** for each source

   (a) **Propagate** sources into the velocity model :$\mathbf{u}_s^i(\mathbf{x}, t)$

   (b) **Back-propagate** the wavefield recorded at receivers :$\mathbf{u}_r^i(\mathbf{x}, T-t)$($T$ is the final time)

3. Cross-correlation of (a) and (b): Imaging condition

Reverse Time Migration Main Steps

1. Get the initial velocity model thanks to seismic acquisition:

   Tomography

2. By using sources and receivers of the seismic acquisition.

   We solve the equation **twice** for each source

   (a) **Propagate** sources into the velocity model :$\mathbf{u}_s^i(\mathbf{x}, t)$

   (b) **Back-propagate** the wavefield recorded at receivers :$\mathbf{u}_r^i(\mathbf{x}, T-t)$($T$ is the final time)

3. Cross-correlation of (a) and (b): Imaging condition

$$\mathbf{I}(\mathbf{x}) = \sum_{i=0}^{nshots} \int_0^T \mathbf{u}_s^i(\mathbf{x}, t) \cdot \mathbf{u}_r^i(\mathbf{x}, T-t) dt$$

Elastic Wave equations

$$\begin{cases} \partial_t \mathbf{u}(\mathbf{x},t) &= \dfrac{1}{\rho(\mathbf{x})} \nabla \cdot \underline{\underline{\sigma}}(\mathbf{x},t) \\ \partial_t \underline{\underline{\sigma}}(\mathbf{x},t) &= \lambda(\mathbf{x}) \nabla \cdot \mathbf{u}(\mathbf{x},t) \mathbf{I}_n + \mu(\mathbf{x})(\nabla \mathbf{u}(\mathbf{x},t) + (\nabla \mathbf{u}(\mathbf{x},t))^T) + f(\mathbf{x},t) \end{cases}$$

Where

- $(\mathbf{x}, t) \in \Omega \times [0, T]$
- ρ is the relative density
- λ is the P-Wave modulus
- μ is the shear modulus
- $\mathbf{I}_n$ is the identity, $n = 2, 3$
- f is a source term

Acoustic wave equations

$$\begin{cases} \rho(\mathbf{x})\dfrac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} & +\nabla p(\mathbf{x},t) & = 0 \\ \dfrac{1}{\mu(\mathbf{x})}\dfrac{\partial p(\mathbf{x},t)}{\partial t} & +\nabla \cdot \mathbf{u}(\mathbf{x},t) & = f(\mathbf{x},t) \end{cases}$$

Where

- $(\mathbf{x}, t) \in \Omega \times [0, T]$
- ρ is the relative density
- μ is the P-Wave modulus
- f is a source term

Computation of the image

Simplest imaging condition :

$$\int_0^T \mathbf{u}_s^i(\mathbf{x}, t) \cdot \mathbf{u}_r^i(\mathbf{x}, t) dt$$

Computation of the image

Simplest imaging condition :

$$\int_0^T \mathbf{u}_s^i(\mathbf{x}, t) \cdot \mathbf{u}_r^i(\mathbf{x}, t) dt$$

$$\sum_{n=0}^{nit} \omega_n \mathbf{u}_s^i(\mathbf{x}, t_n) \cdot \mathbf{u}_r^i(\mathbf{x}, t_n)$$

Consequences

- Is it relevant to use elastic waves?
- How to optimize the occupation of the memory?

Store everything ?

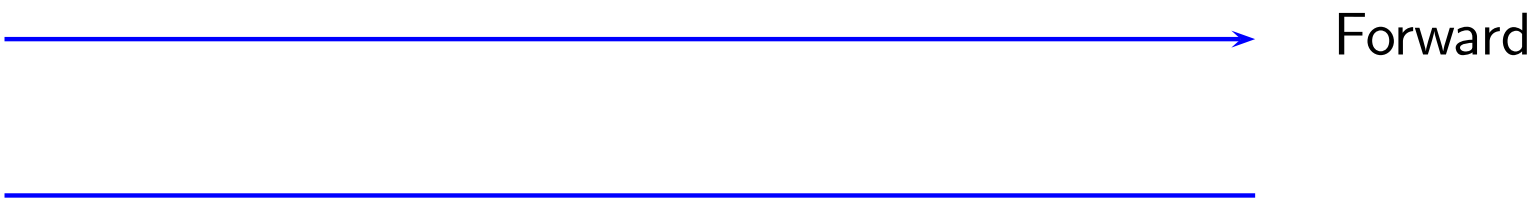
Storage

Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.

Store everything ?

Storage

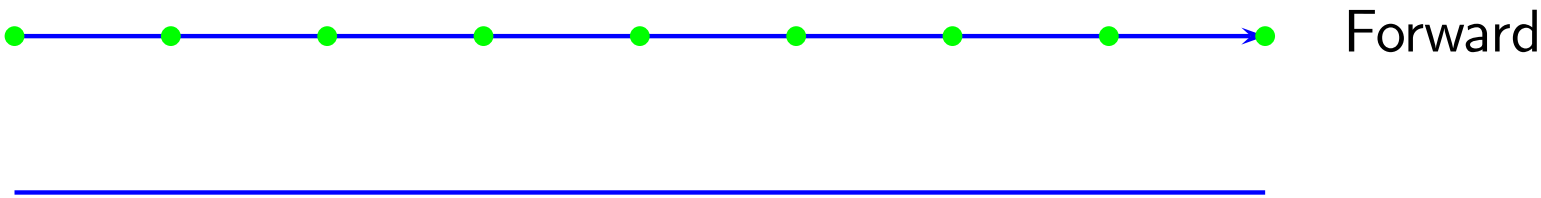
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

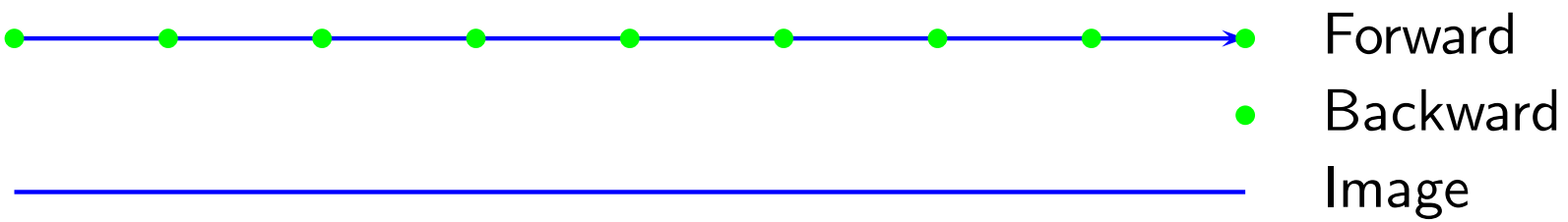
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

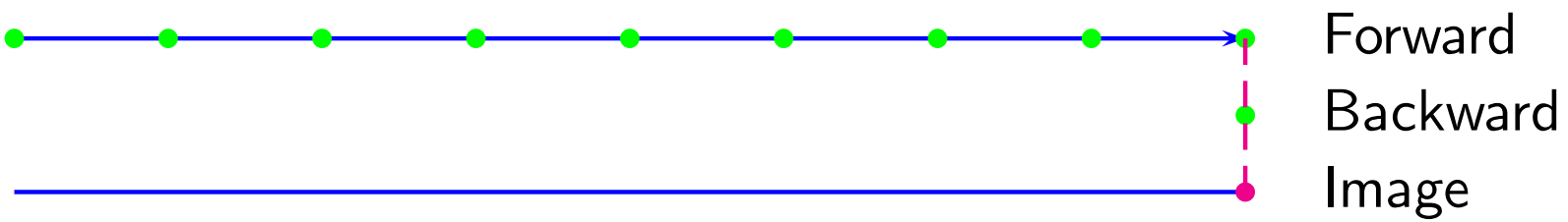
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

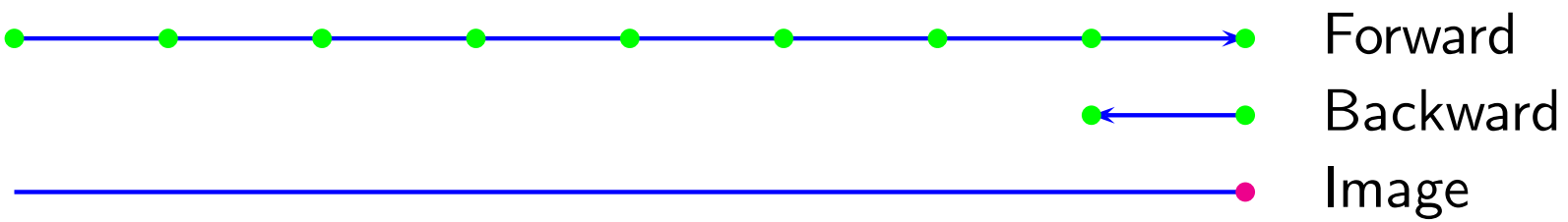
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

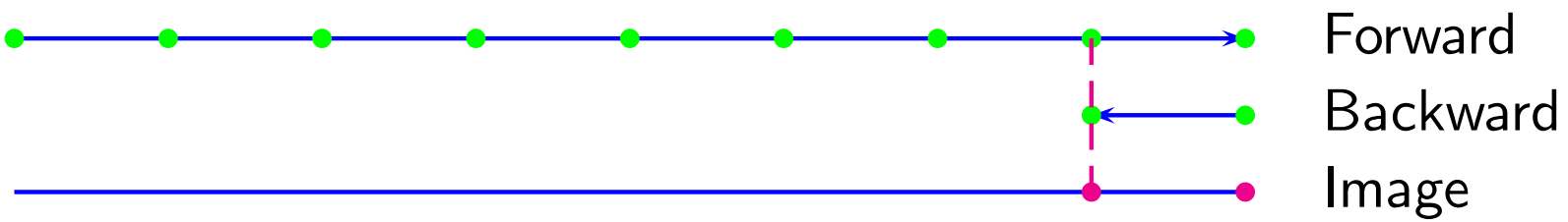
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

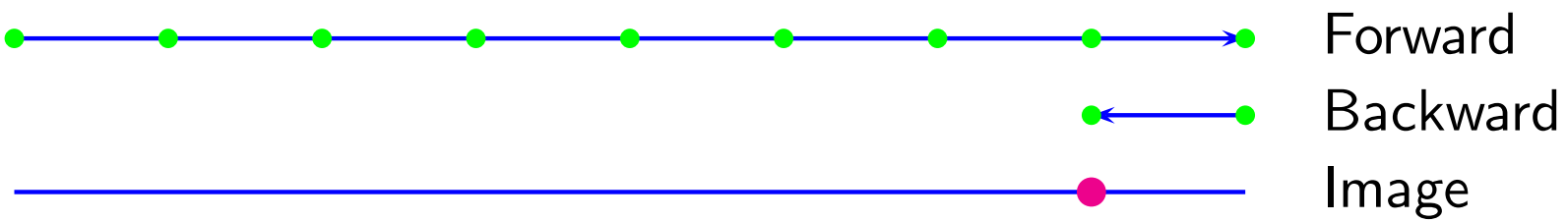
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

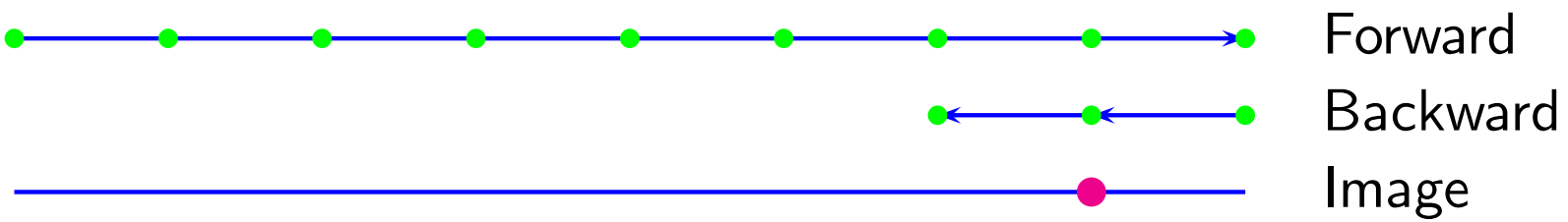
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

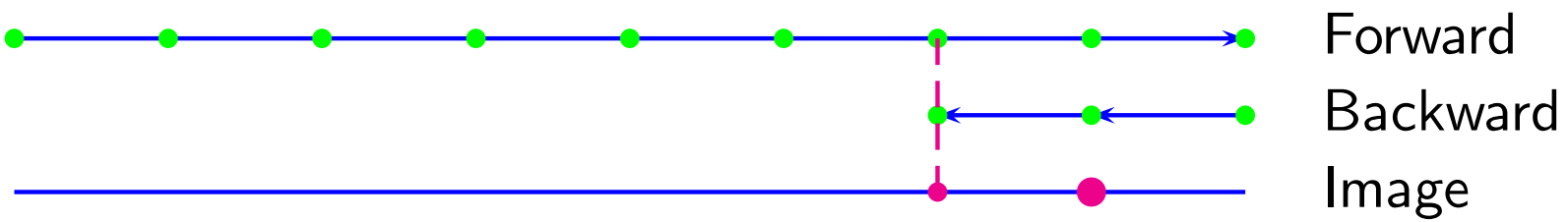
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

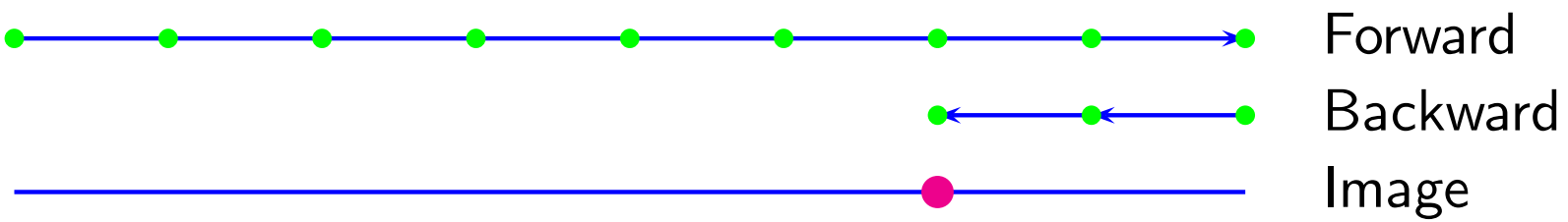
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

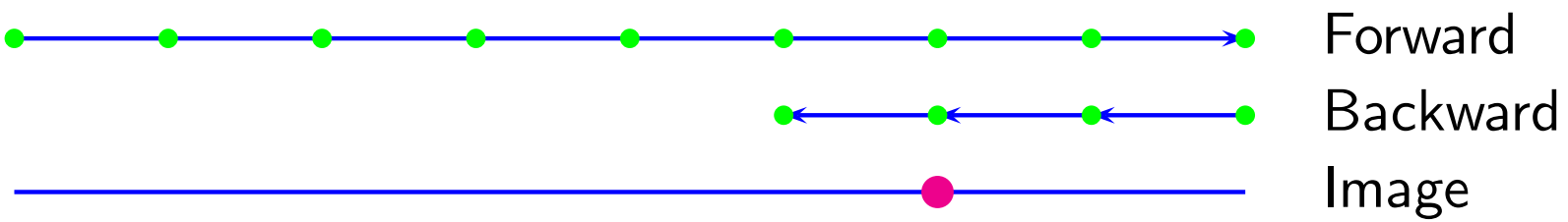
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

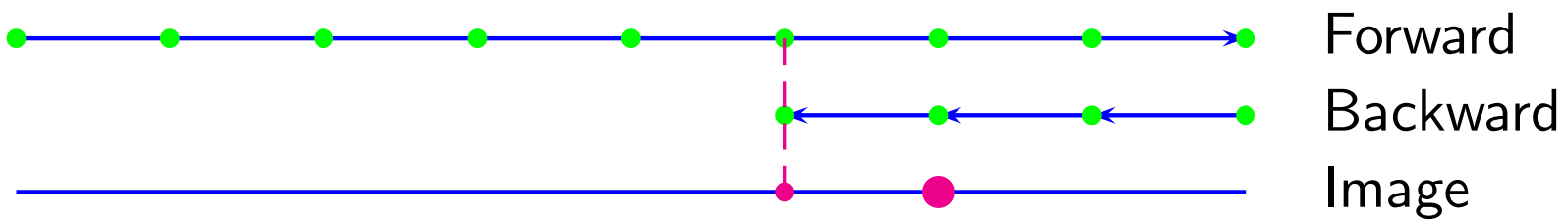
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

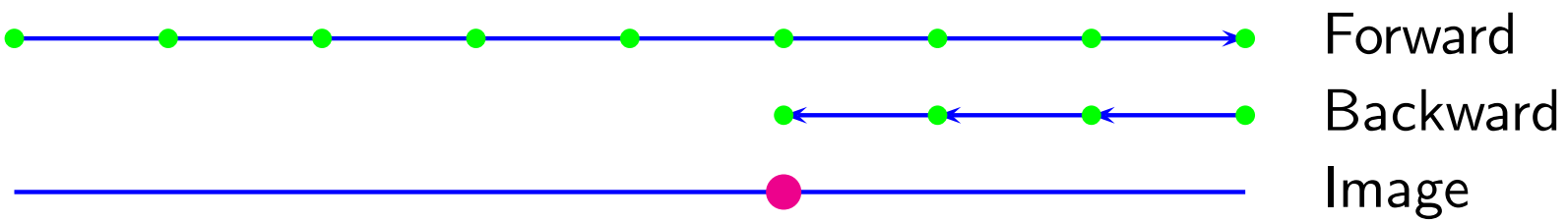
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

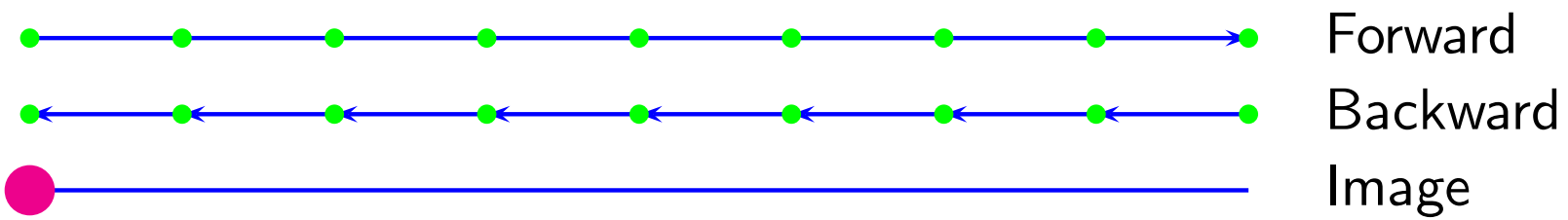
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

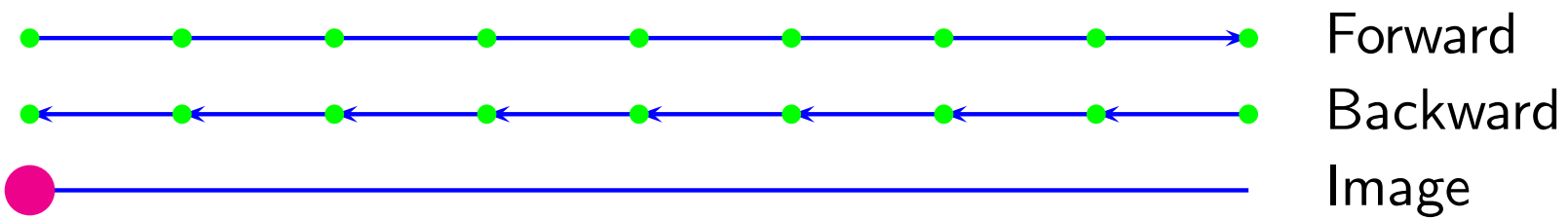
Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Store everything ?

Storage

Record the entire state $\mathbf{u}_s^i(t_n), n = 0, ..., N-1$ and access it when needed.



Computational Cost $\mathcal{O}(N)$, storage Cost $\mathcal{O}(N)$

Recompute each time step ?

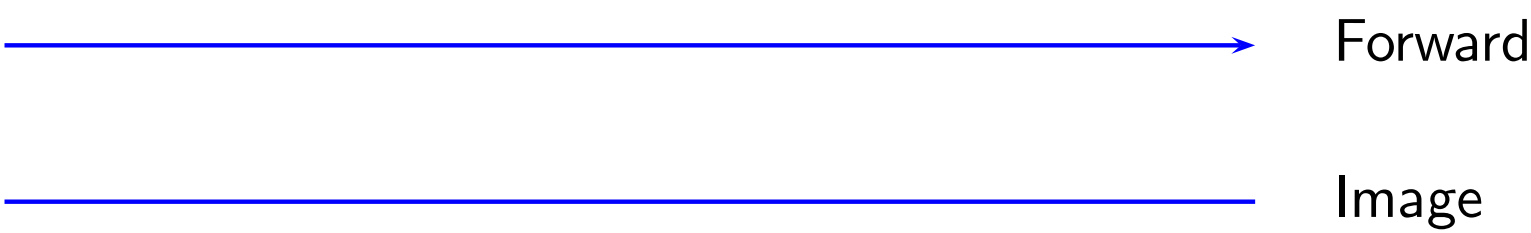
Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$

Recompute each time step ?

Compute each step

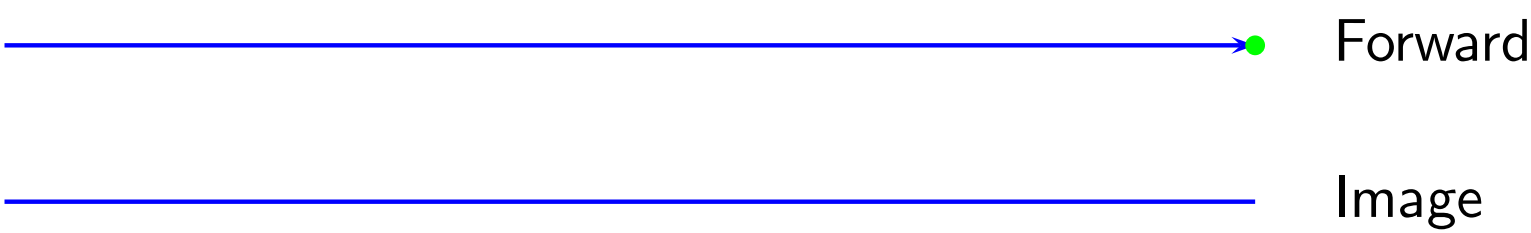
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

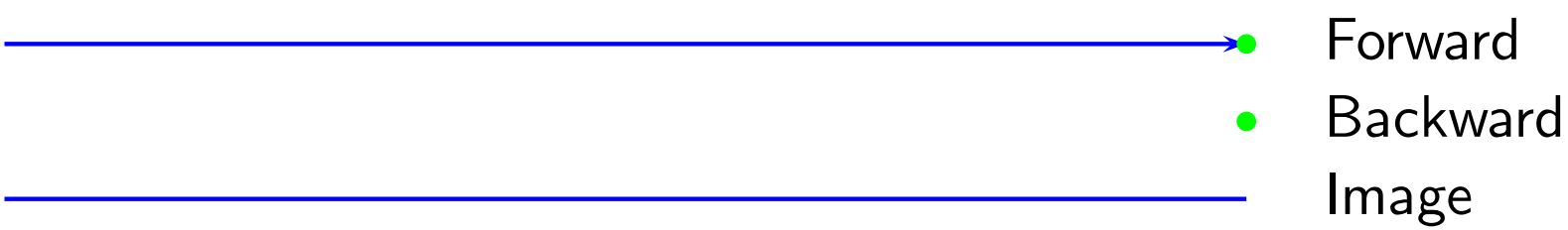
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

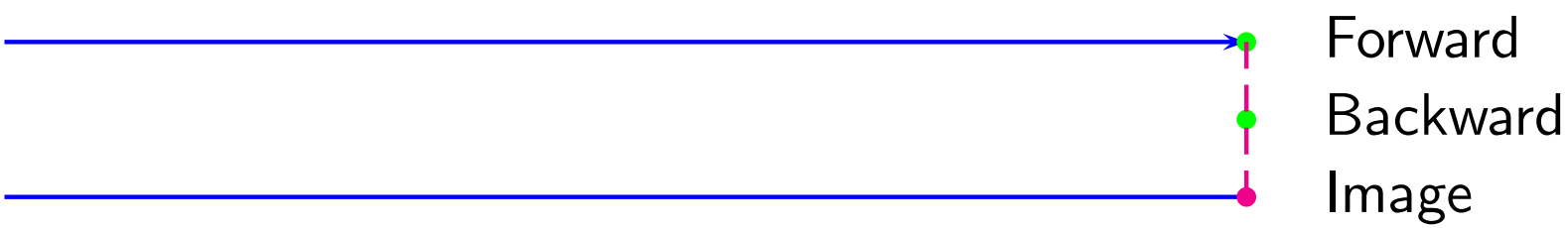
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$

Recompute each time step ?

Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$

Recompute each time step ?

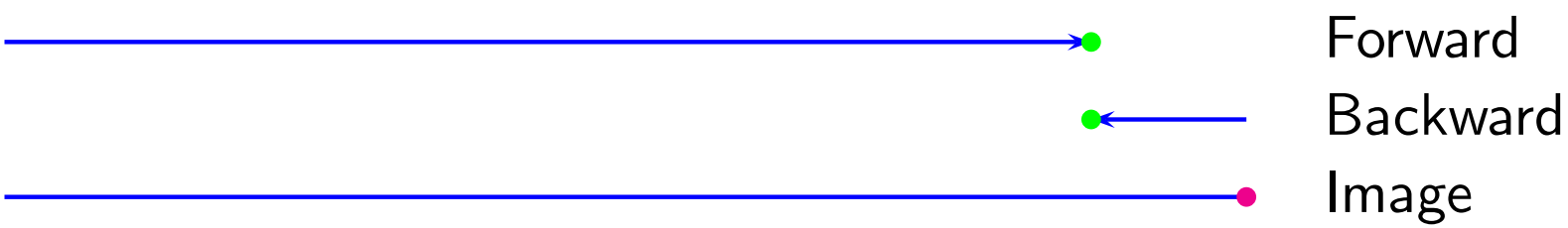
Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$

Recompute each time step ?

Compute each step

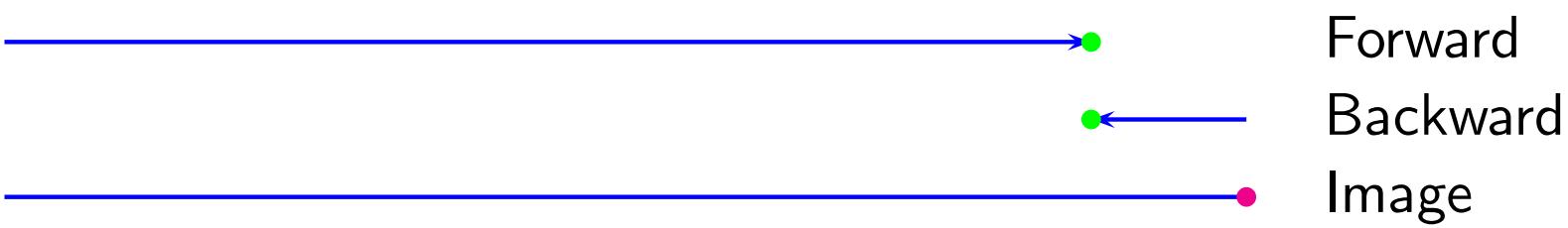
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

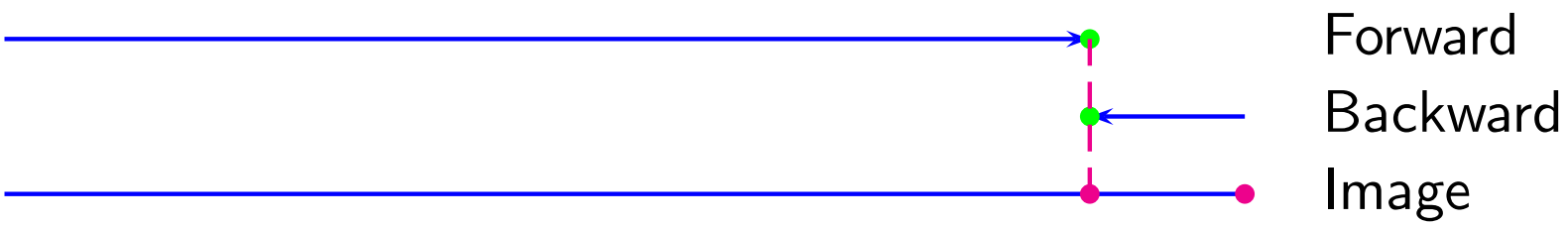
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

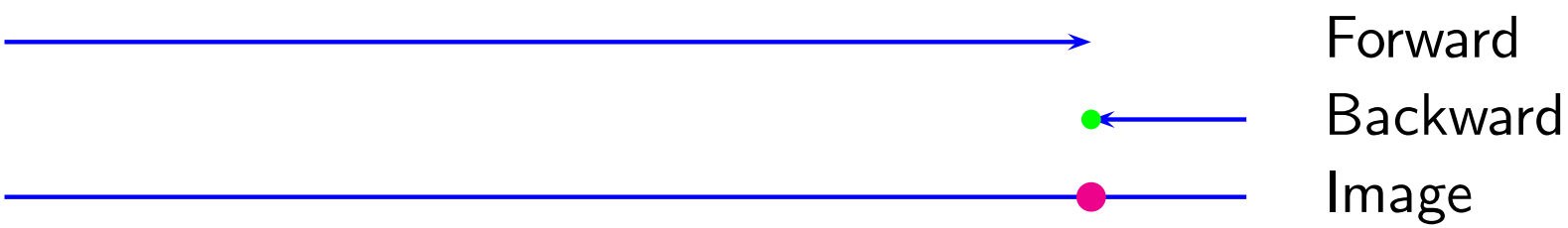
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

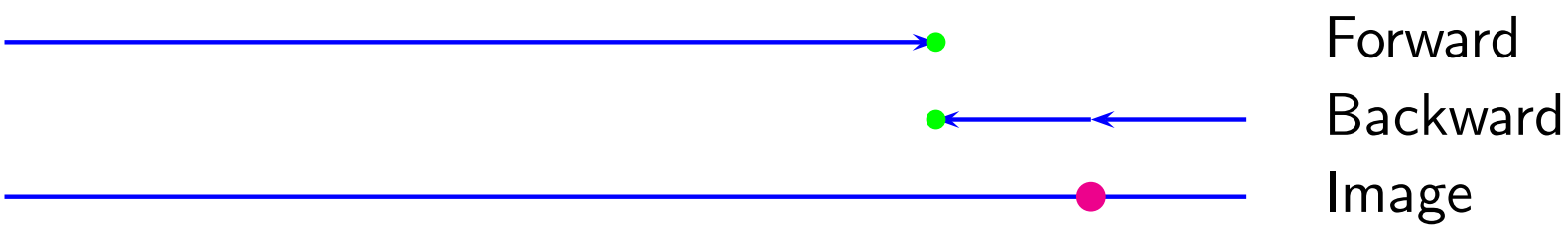
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

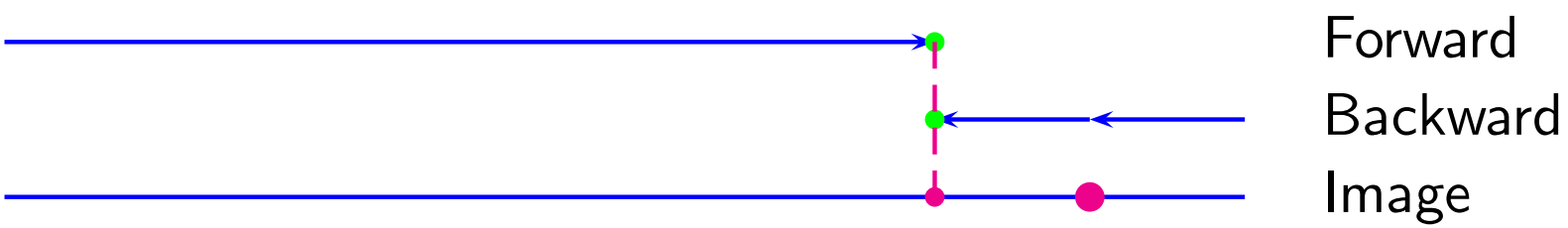
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

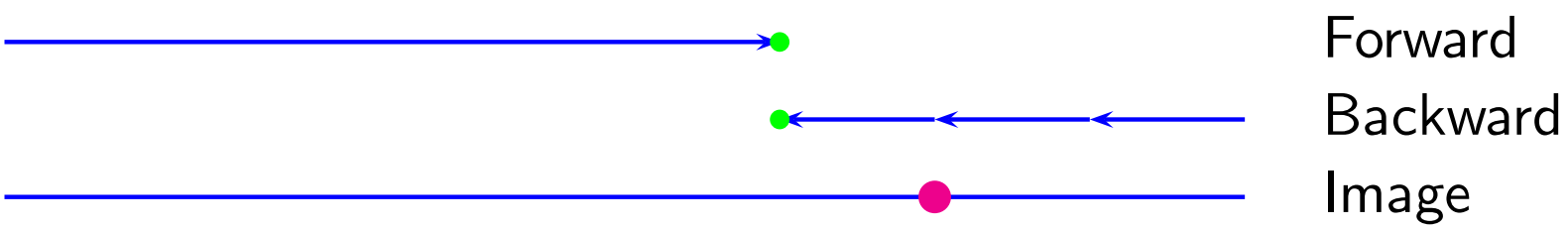
Compute each step

Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$

Recompute each time step ?

Compute each step

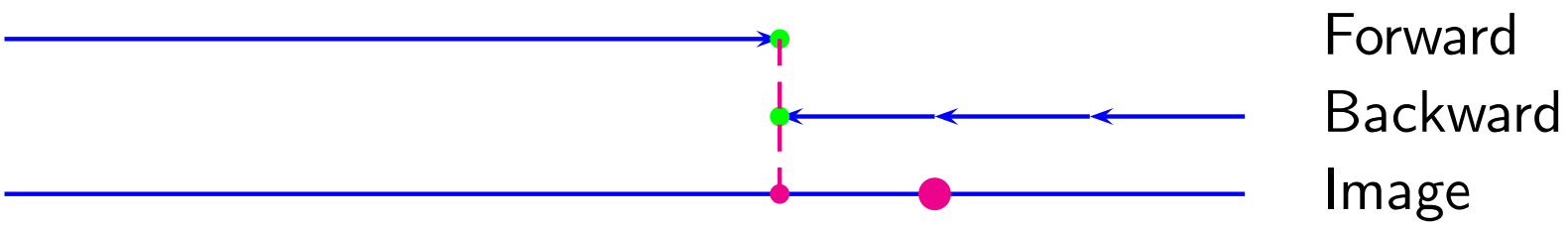
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

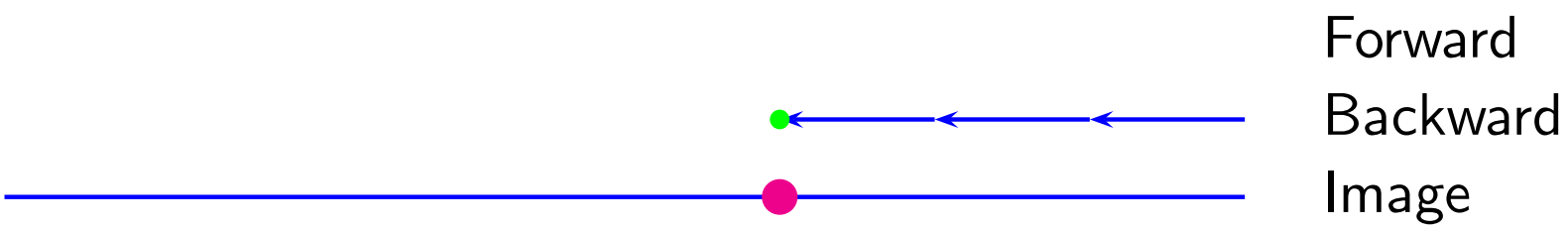
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

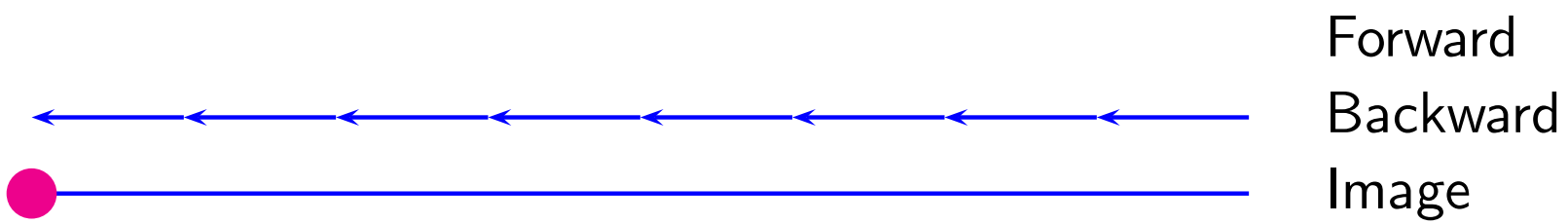
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

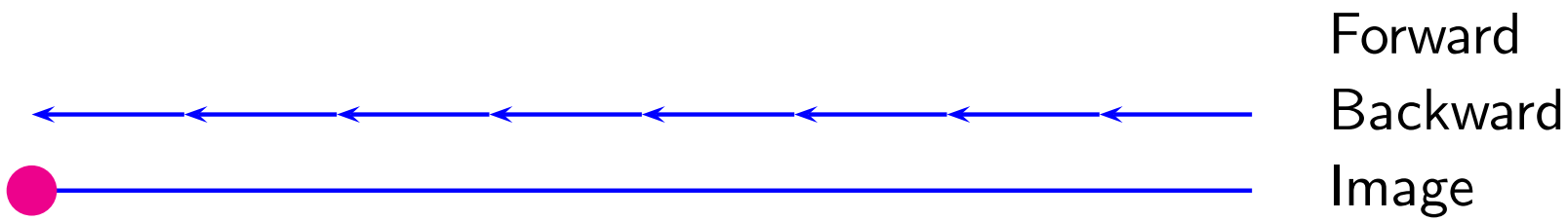
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Recompute each time step ?

Compute each step

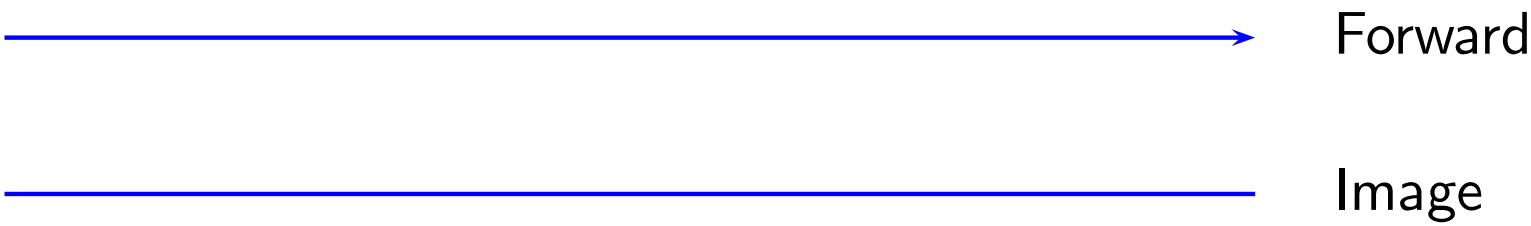
Compute $\mathbf{u}_s^i(t_n)$ for each $n = N-1, ..., 0$



Computational Cost $\mathcal{O}(N^2)$, Storage Cost $\mathcal{O}(1)$

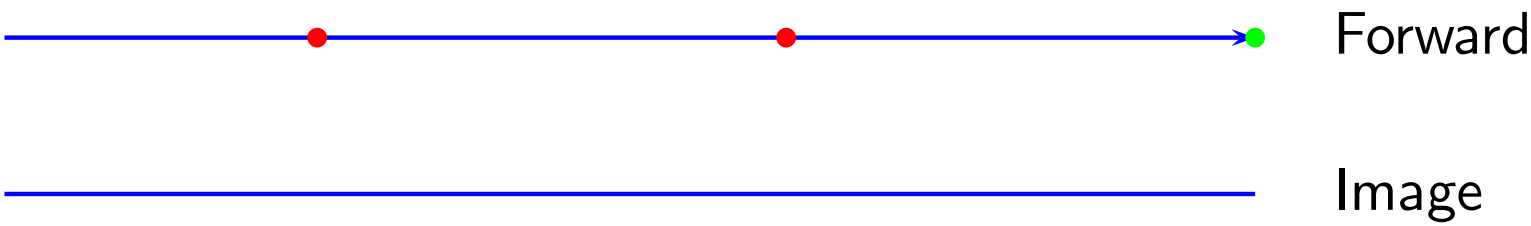
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



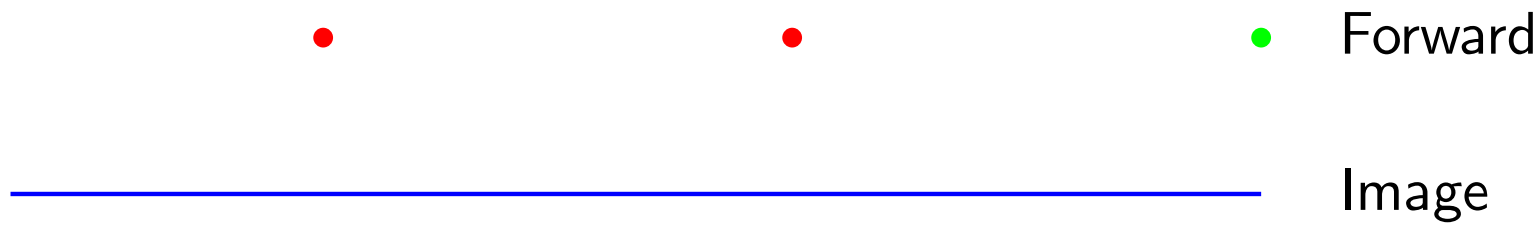
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



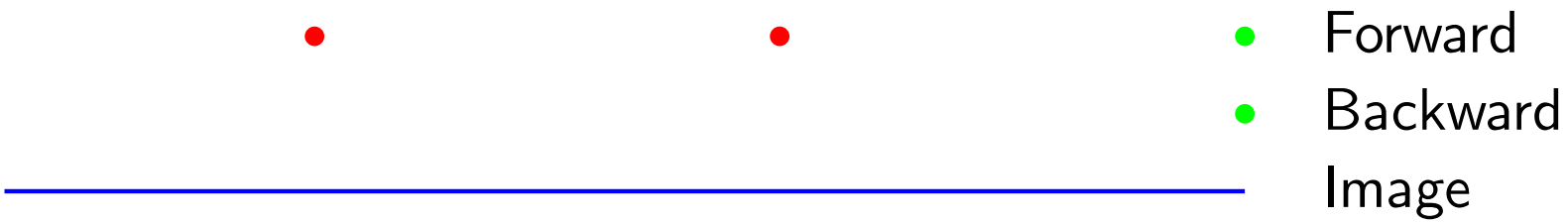
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



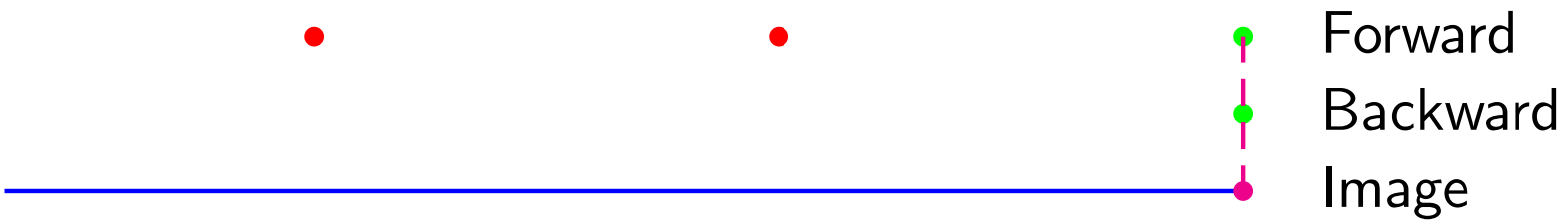
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



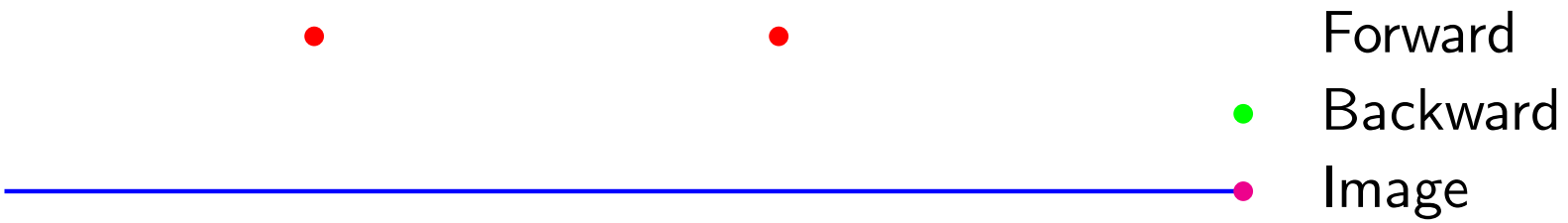
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



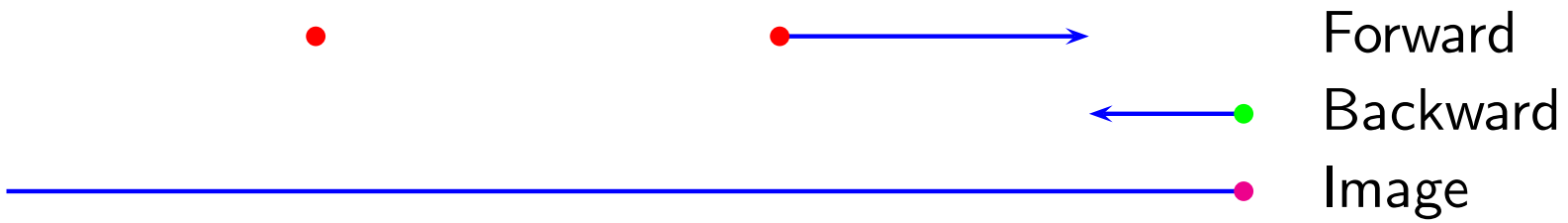
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



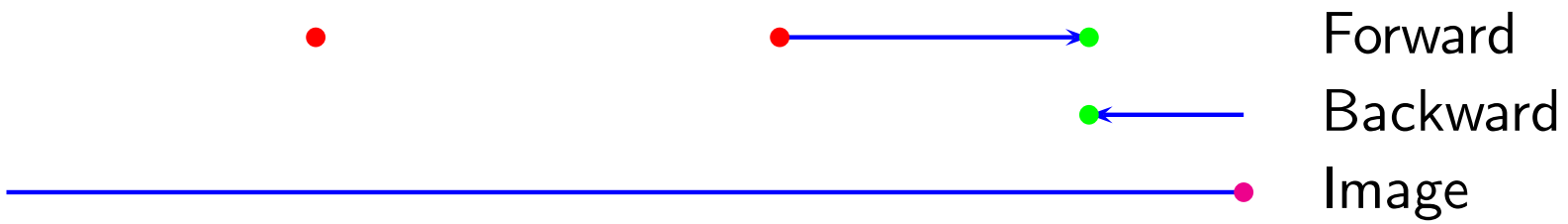
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



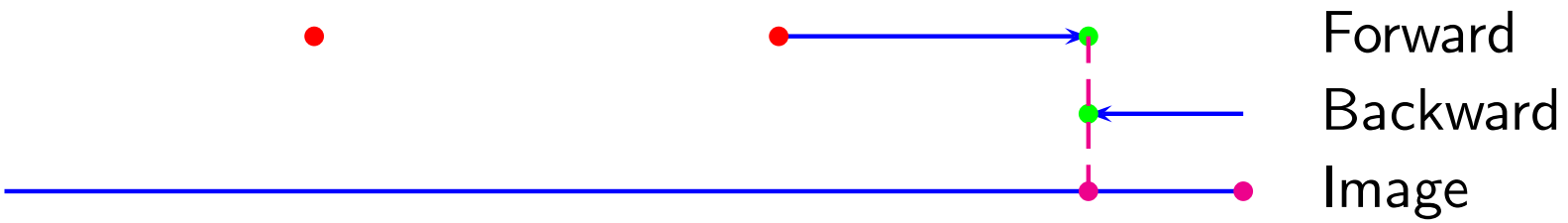
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



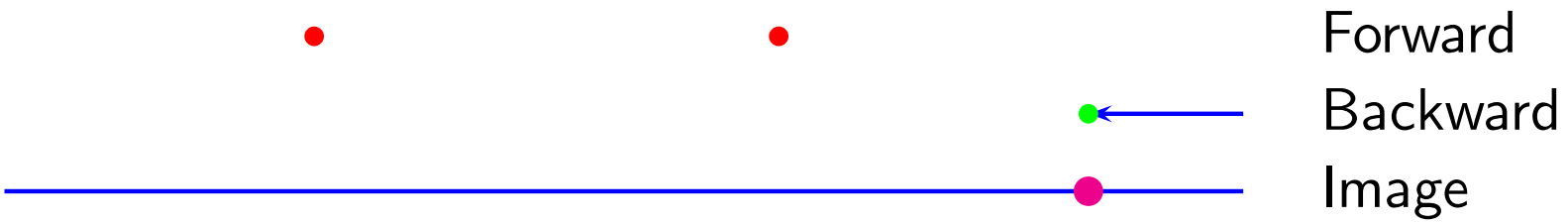
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



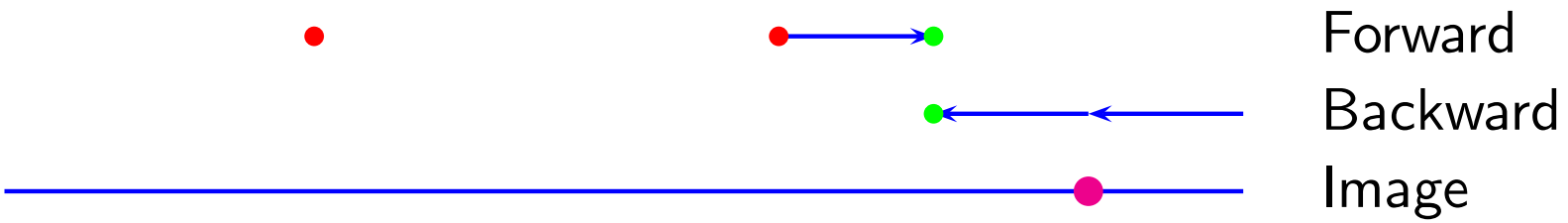
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



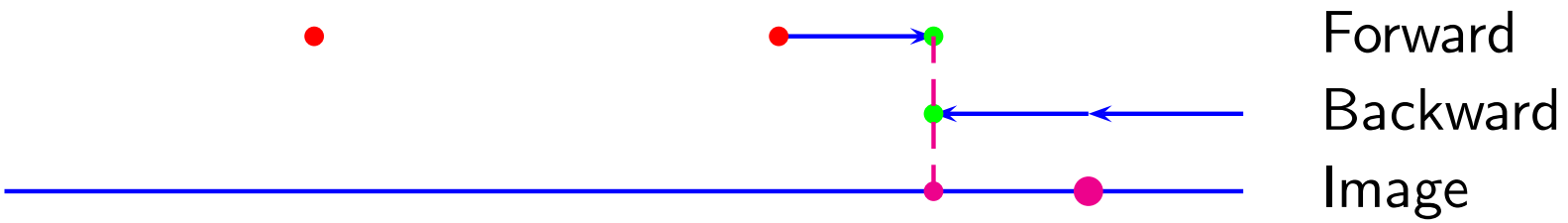
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



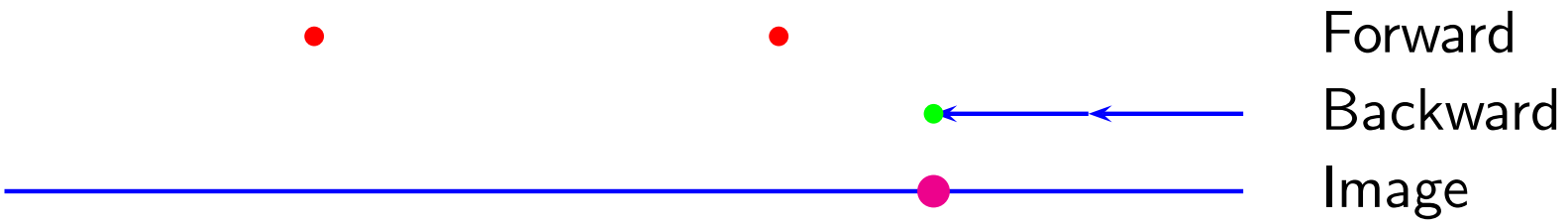
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



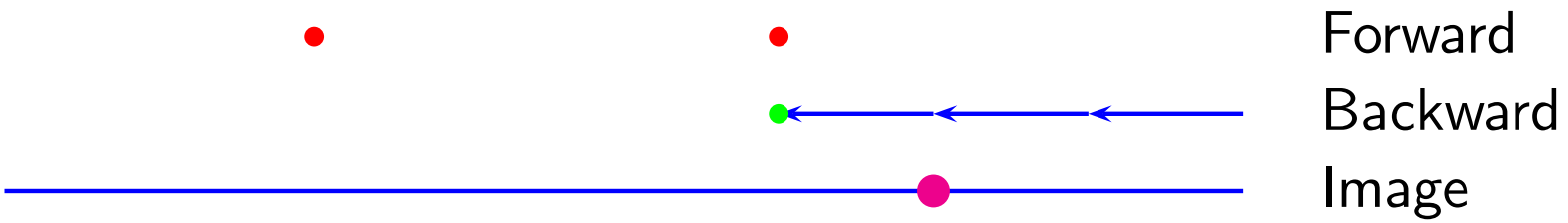
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



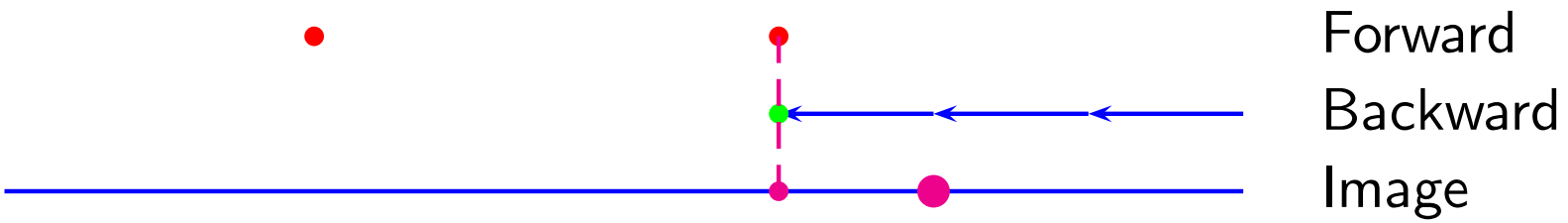
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



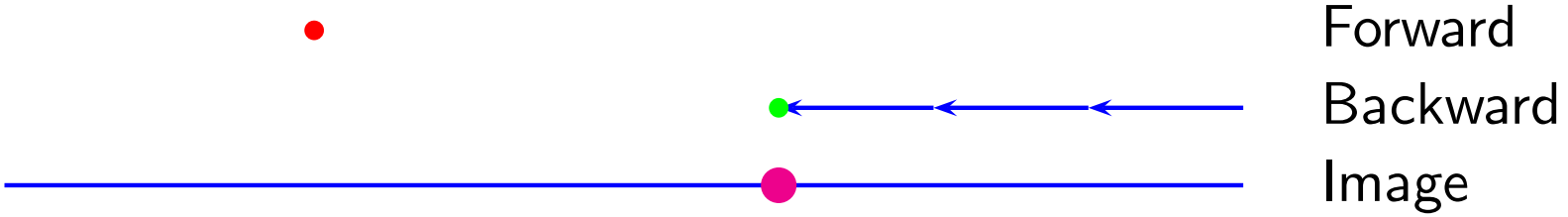
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



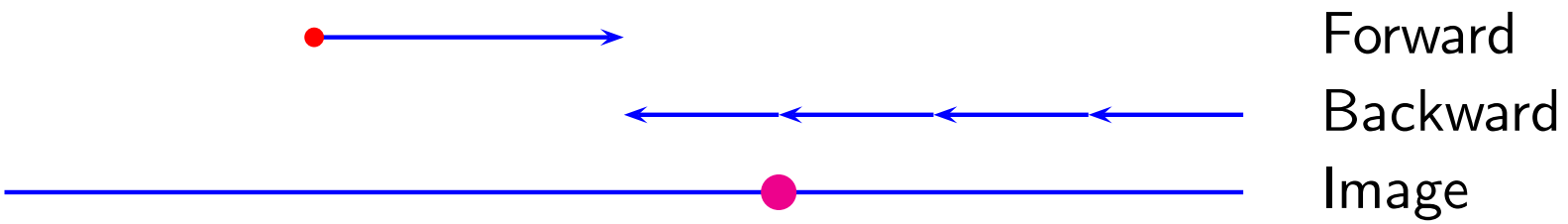
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



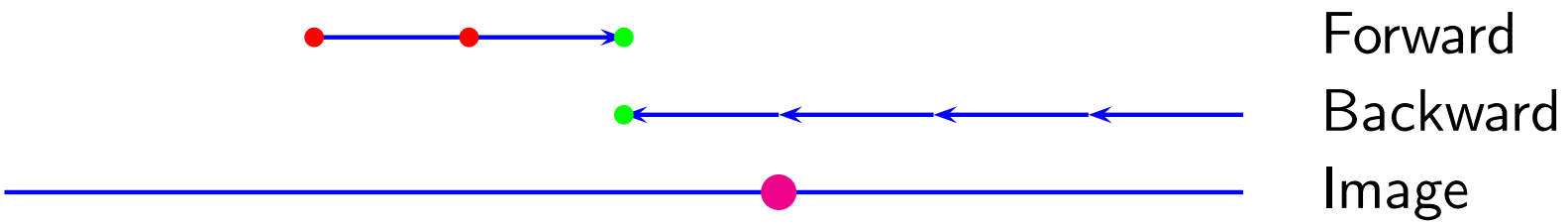
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



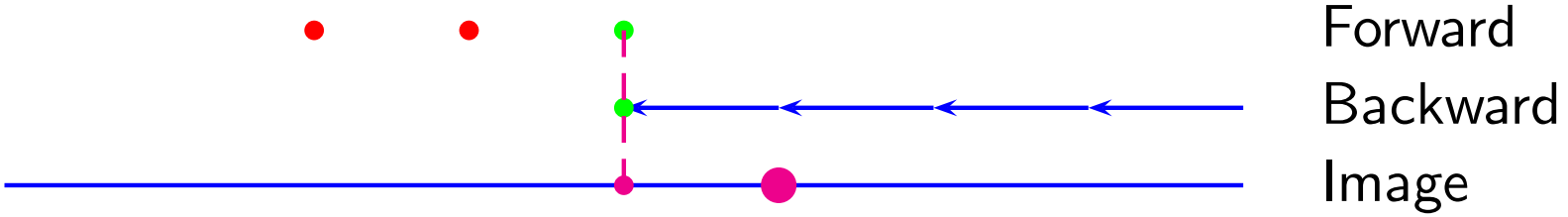
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



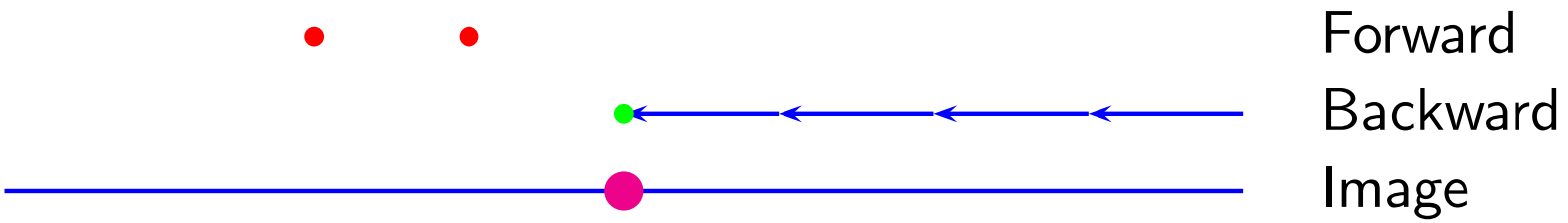
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



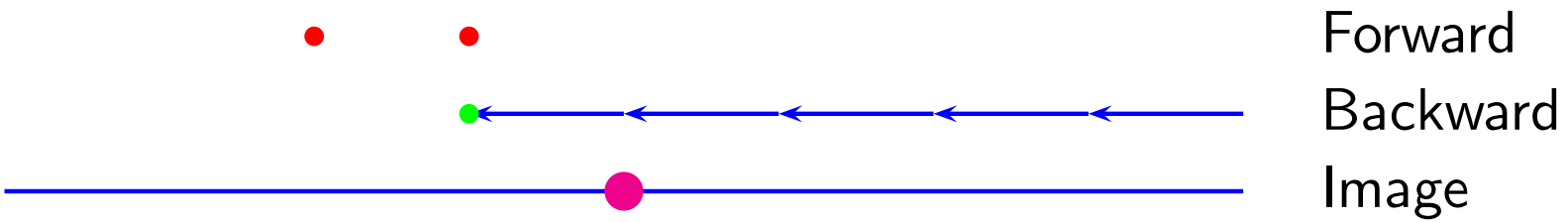
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



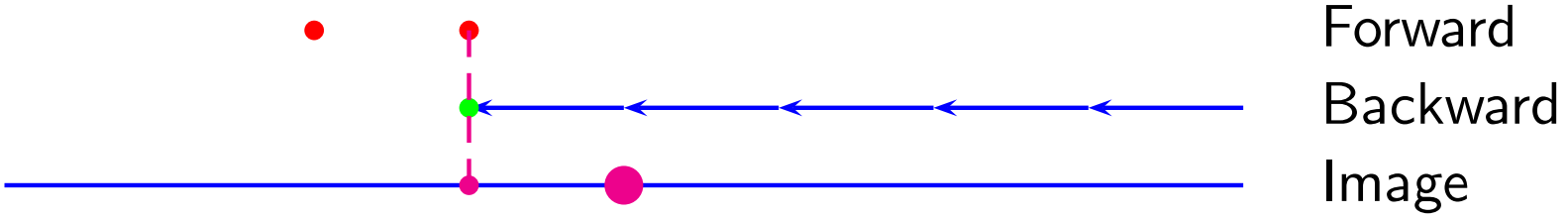
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



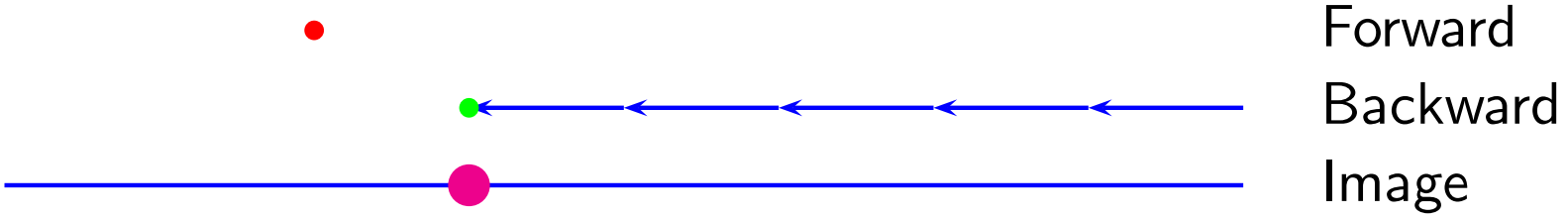
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



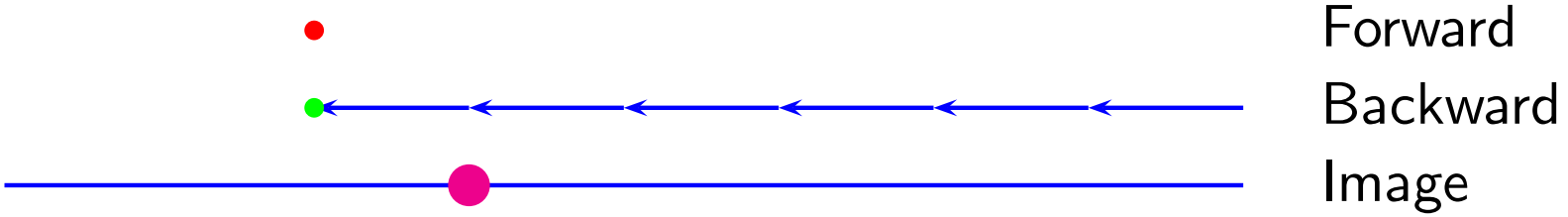
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



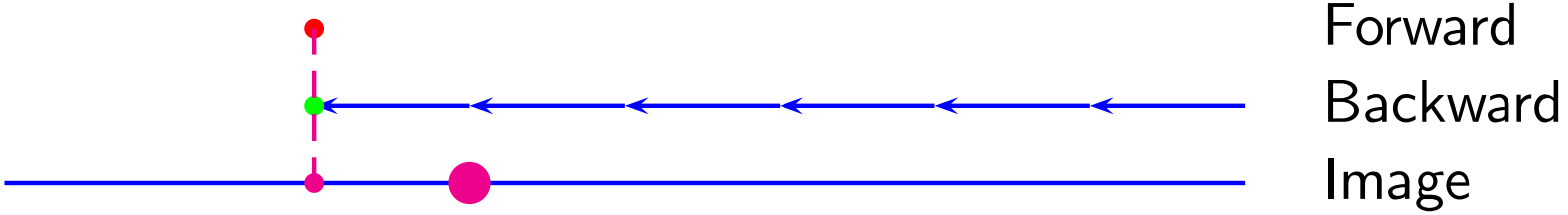
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



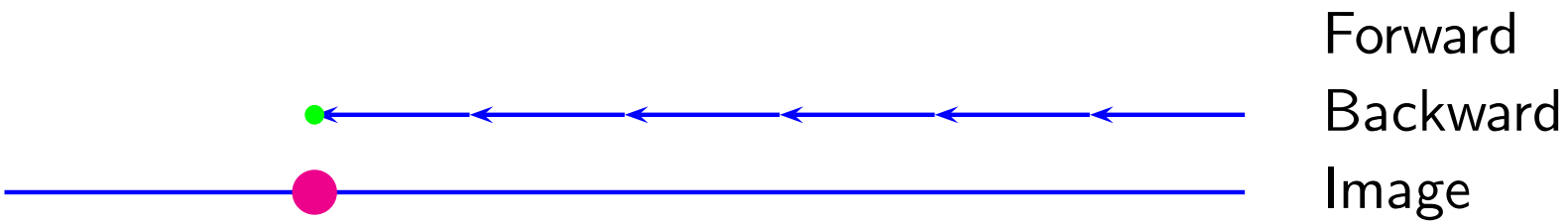
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



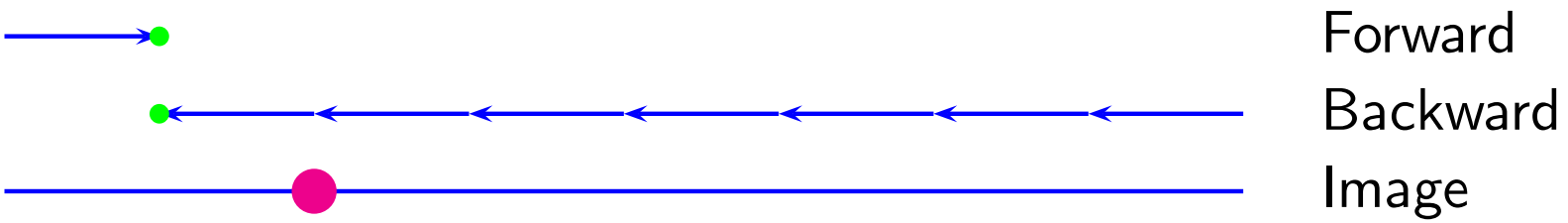
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



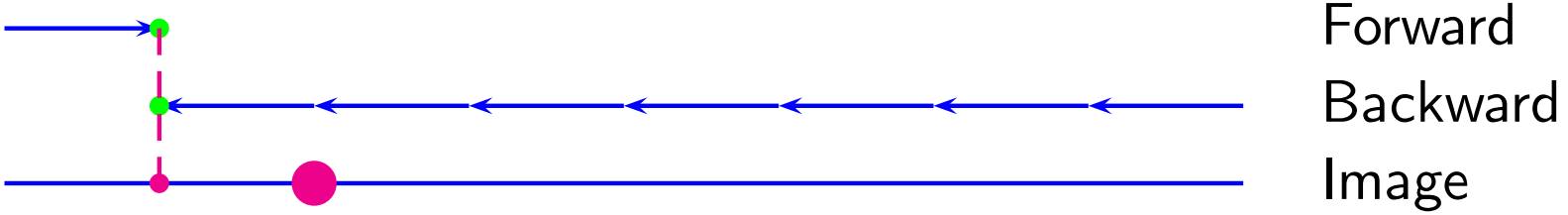
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



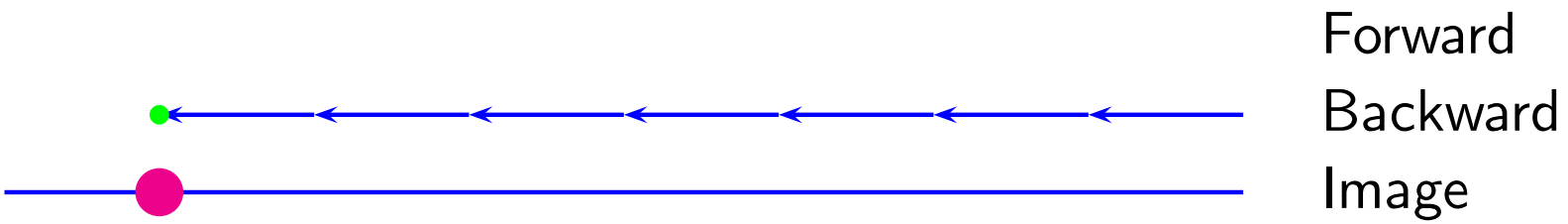
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



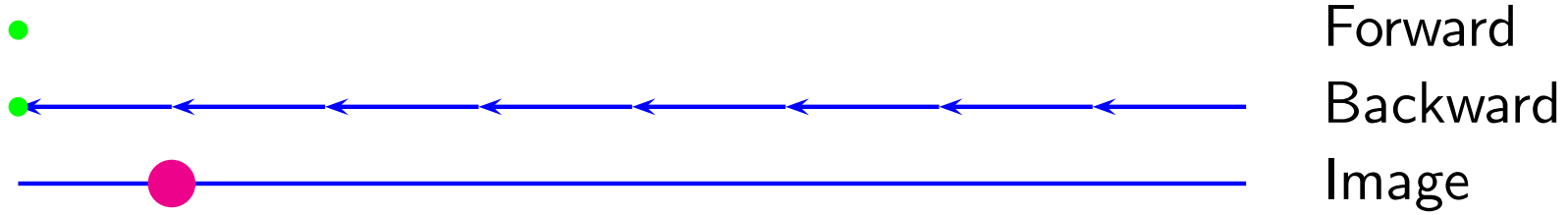
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



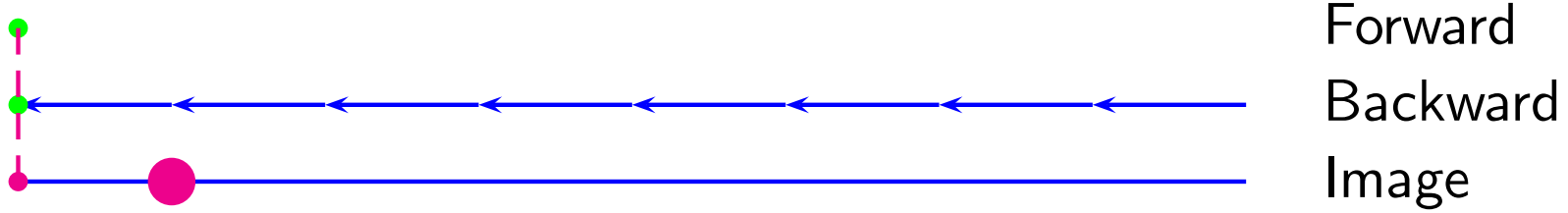
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



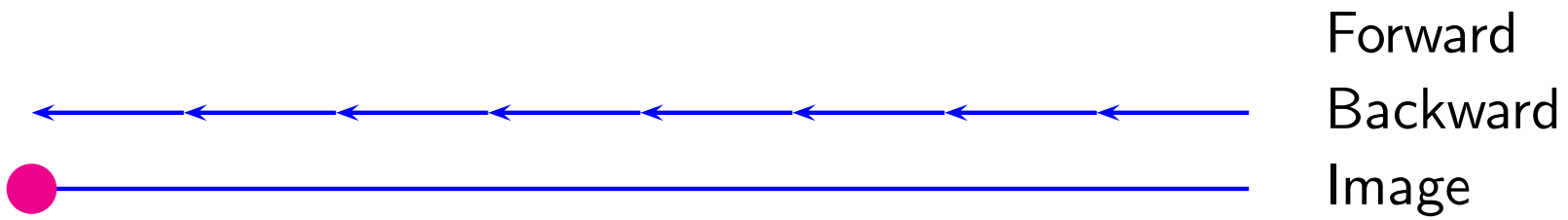
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



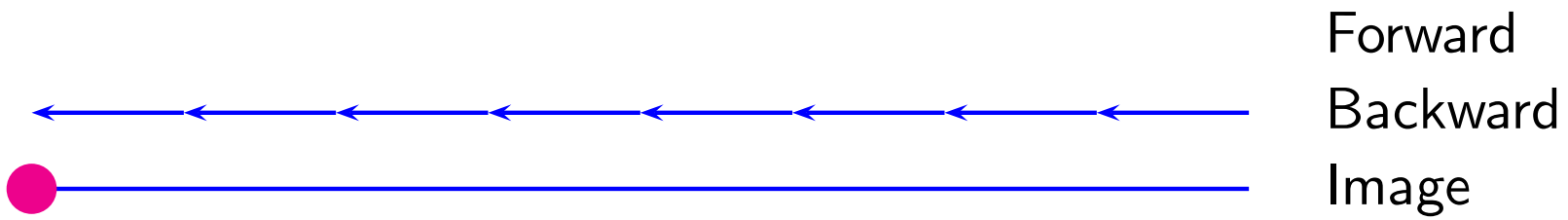
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



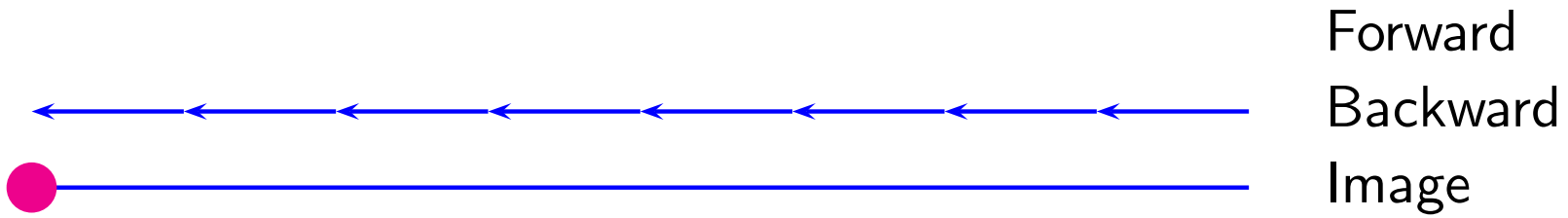
Griewank

- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



Griewank

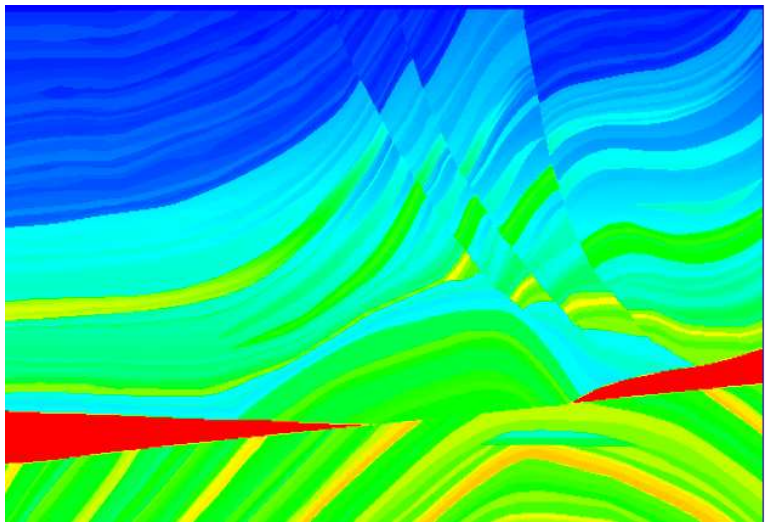
- During the **forward** : Store only a few time steps $\mathbf{u}_s^i(t_n)$ in buffers called checkpoints
- During the **backward** : Calculate $\mathbf{u}_s^i(t_n)$ several times from the last checkpoint.



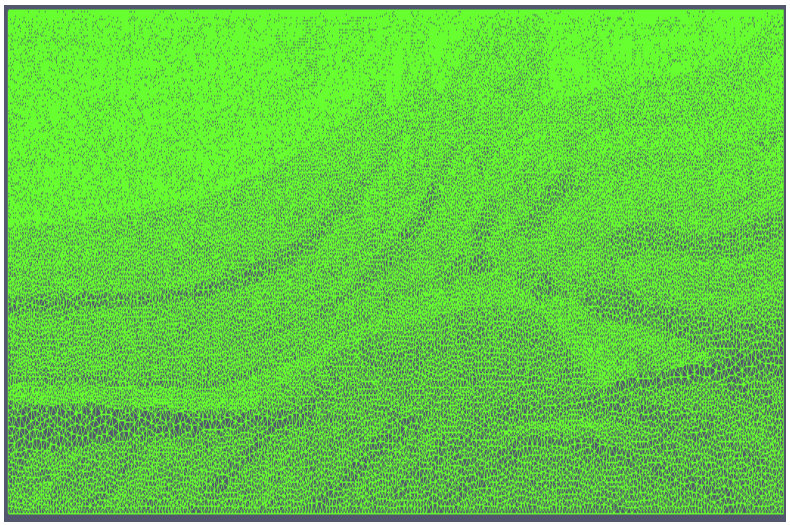
Computational Cost $\mathcal{O}(N)$, Storage Cost $\mathcal{O}(N_B)$

Analysis of strategies	Strategy 1	Strategy 2	Griewank
CPU time	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$	$\mathcal{O}(N)$
Storage Cost	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\mathcal{O}(N_B)$

Marmousi parameters



Velocity Model



Mesh : 21469 triangles

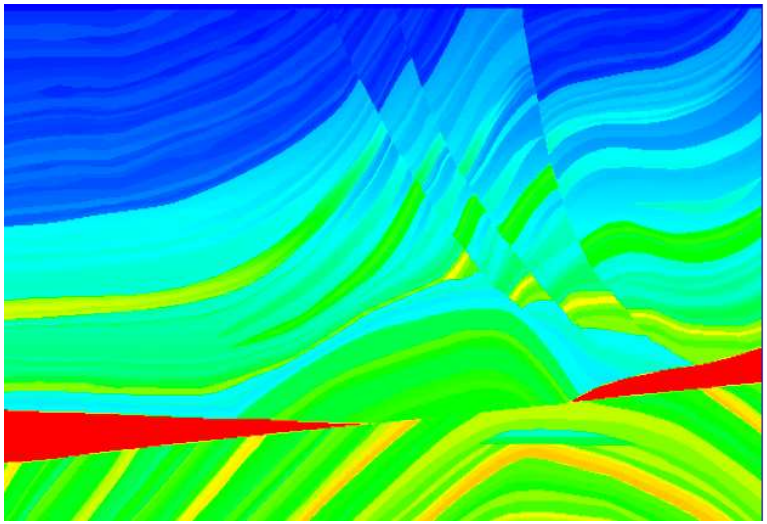
Marmousi parameters

- Dimension : 9500m $\times$ 3000m
- $V_{p_{min}} = 1500\text{m.s}^{-1}$, $V_{p_{max}} = 5500\text{m.s}^{-1}$
- Number of shots : 240
- Number of receivers : 96
- Time of the experience : 6 sec
- Claerbout Imaging Condition

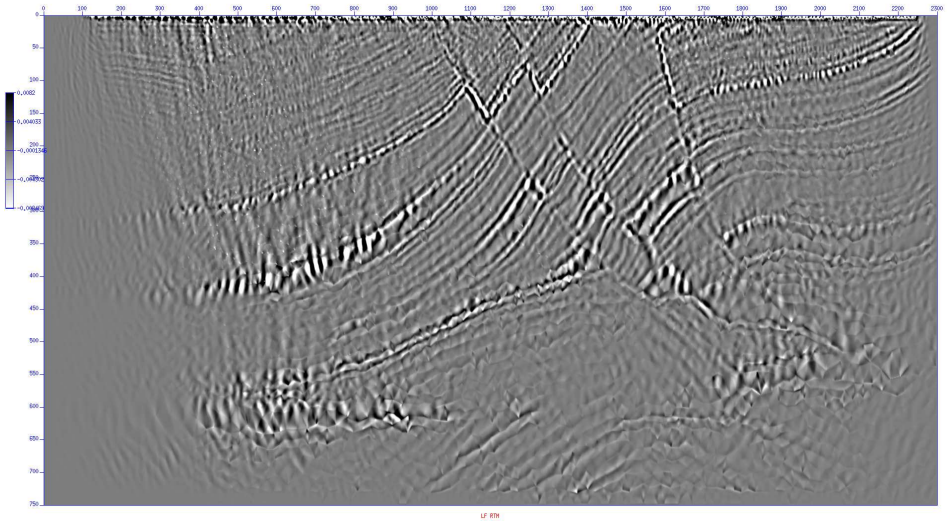
$$I(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T \mathbf{U}_i^s(x,y,t)\mathbf{U}_r^i(x,y,t)$$

- Storage Strategy: Griewank algorithm

Marmousi Acoustic



Velocity Model



RTM acoustic

Analysis of strategies	Strategy 1	Strategy 2	Griewank
CPU time	20 min	105 days	21 min
Storage Cost	1000 Mo	4 Mo	40 Mo

Use of elastic Waves

- More accurate modeling
- Two types of waves : P waves and S waves
- How can we use this different wave fields ?

Separate the Waves

- Yan & Sava Condition

J. Yan and P. Sava (2008)
Isotropic angle-domain elastic reverse-time migration,
Geophysics, 73 , no. 6, S229-S239.

Separate the Waves

- Yan & Sava Condition
  - $I_{PP}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \cdot \mathbf{u}_s^i(\mathbf{x}, t)][\nabla \cdot \mathbf{u}_r^i(\mathbf{x}, t)]dt$

J. Yan and P. Sava (2008)
Isotropic angle-domain elastic reverse-time migration,
Geophysics, 73 , no. 6, S229-S239.

Separate the Waves

- Yan & Sava Condition
  - $I_{PP}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \cdot \mathbf{u}_s^i(\mathbf{x}, t)][\nabla \cdot \mathbf{u}_r^i(\mathbf{x}, t)]dt$
  - $I_{SS}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \times \mathbf{u}_s^i(\mathbf{x}, t)] \cdot [\nabla \times \mathbf{u}_r^i(\mathbf{x}, t)]dt$

J. Yan and P. Sava (2008)
Isotropic angle-domain elastic reverse-time migration,
Geophysics, 73 , no. 6, S229-S239.

Separate the Waves

- Yan & Sava Condition
  - $I_{PP}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \cdot \mathbf{u}_s^i(\mathbf{x}, t)][\nabla \cdot \mathbf{u}_r^i(\mathbf{x}, t)]dt \Rightarrow P$ Wave
  - $I_{SS}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \times \mathbf{u}_s^i(\mathbf{x}, t)] \cdot [\nabla \times \mathbf{u}_r^i(\mathbf{x}, t)]dt$

J. Yan and P. Sava (2008)
Isotropic angle-domain elastic reverse-time migration,
Geophysics, 73 , no. 6, S229-S239.

Separate the Waves

- Yan & Sava Condition
  - $I_{PP}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \cdot \mathbf{u}_s^i(\mathbf{x}, t)][\nabla \cdot \mathbf{u}_r^i(\mathbf{x}, t)]dt \Rightarrow P$ Wave
  - $I_{SS}(\mathbf{x}) = \sum_{i=1}^{n} \int_0^T [\nabla \times \mathbf{u}_s^i(\mathbf{x}, t)] \cdot [\nabla \times \mathbf{u}_r^i(\mathbf{x}, t)]dt \Rightarrow S$ Wave

J. Yan and P. Sava (2008)
Isotropic angle-domain elastic reverse-time migration,
Geophysics, 73 , no. 6, S229-S239.

Kernels

Define three Kernels associated to the elastic parameters ρ, μ and κ :

$$K_\rho(\mathbf{x}), K_\mu(\mathbf{x}), K_\kappa(\mathbf{x})$$

A. Tarantola (2005)

Inverse problem theory and methods for model parameter estimation, Society for Industrial and Applied Mathematics.

H. Zhu, Y. Luo, T. Nissen-Meyer, C. Morency, and J. Tromp (2009)

Elastic imaging and time-lapse migration based on adjoints methods, Geophysics, 74 , no. 6, S167-S177.

Kernels

Define three Kernels associated to the elastic parameters ρ, μ and κ :

$$K_\rho(\mathbf{x}), K_\mu(\mathbf{x}), K_\kappa(\mathbf{x})$$

$$K(\mathbf{x}) = K_\rho(\mathbf{x}) + K_\mu(\mathbf{x}) + K_\kappa(\mathbf{x})$$

Tromp, J., C. H. Tape, and Q. Liu (2005)
Seismic tomography, adjoint methods, time reversal, and banana-doughnut kernels,
Geophysics J. Int. 160, 195–216.

Kernel ρ

$$K_\rho(\mathbf{x}) = \sum_{i=1}^{N} \int_0^T \rho(\mathbf{x}) \mathbf{u}_s^i(\mathbf{x}, t) \cdot \mathbf{u}_r^i(\mathbf{x}, t) dt$$

Kernel κ

$$K_\kappa(\mathbf{x}) = -\sum_{i=1}^{n} \int_0^T \kappa(\mathbf{x}) \left[\nabla \cdot \mathbf{U}_r^i(\mathbf{x}, t)\right] \left[\nabla \cdot \mathbf{U}_s^i(\mathbf{x}, t)\right] dt$$

With :

$$\mathbf{U}_r^i(\mathbf{x}, t) = \int_0^t \mathbf{u}_r^i(\mathbf{x}, \tau) d\tau$$

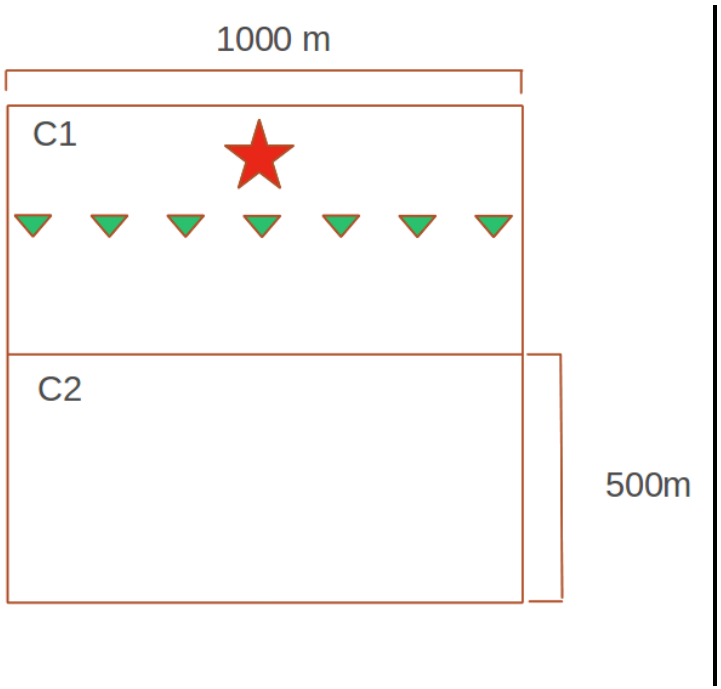
$$\mathbf{U}_s^i(\mathbf{x}, t) = \int_0^t \mathbf{u}_s^i(\mathbf{x}, \tau) d\tau$$

Kernel μ

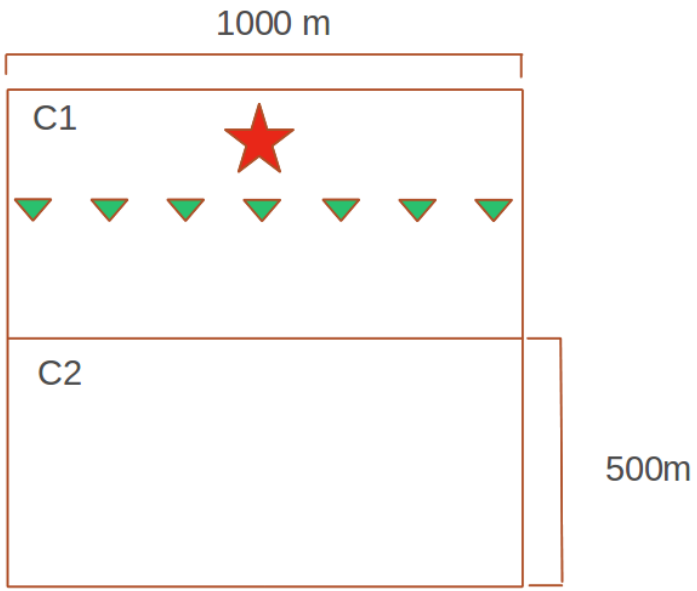
In $2D$, we obtain

$$K_\mu(\mathbf{x}) = -\int_0^T \mu(\mathbf{x}) \Bigg((\nabla \times \mathbf{u}_s^i)(\nabla \times \mathbf{u}_r^i) + (\nabla \cdot \mathbf{u}_s^i)(\nabla \cdot \mathbf{u}_r^i) - 2 \begin{pmatrix} \partial_x \mathbf{u}_{sx}^i \\ \partial_y \mathbf{u}_{sy}^i \end{pmatrix} \cdot \begin{pmatrix} \partial_y \mathbf{u}_{ry}^i \\ \partial_x \mathbf{u}_{rx}^i \end{pmatrix} \Bigg) dt$$

2D Square test case



2D Square test case



- Square 1000m $\times$ 1000m
- Simulation time : 1 s
- Dirichlet's condition on top
- Boundary $y = 500m$
- $V_p = 2500\_5000\text{m.s}^{-1}$, $V_s = 1250\_2500\text{m.s}^{-1}$
- Ricker source term : 20Hz
- 30 sources and 30 receivers

Classical condition

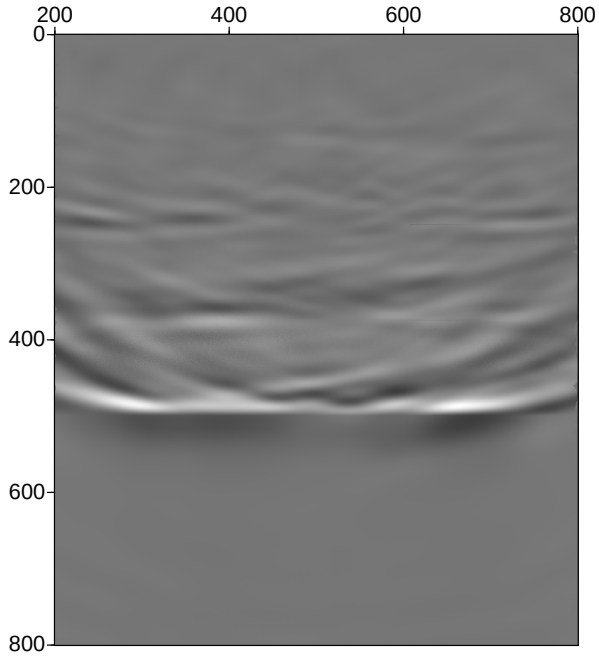
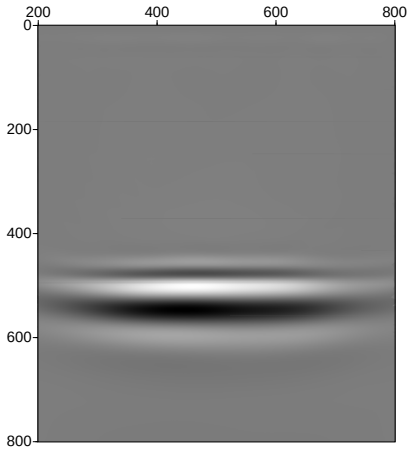
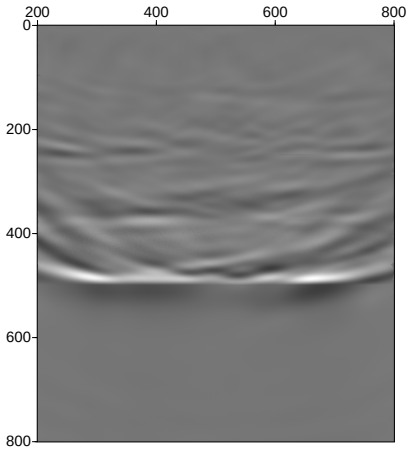


Figure: Classical Condition

Acoustic vs. elastic

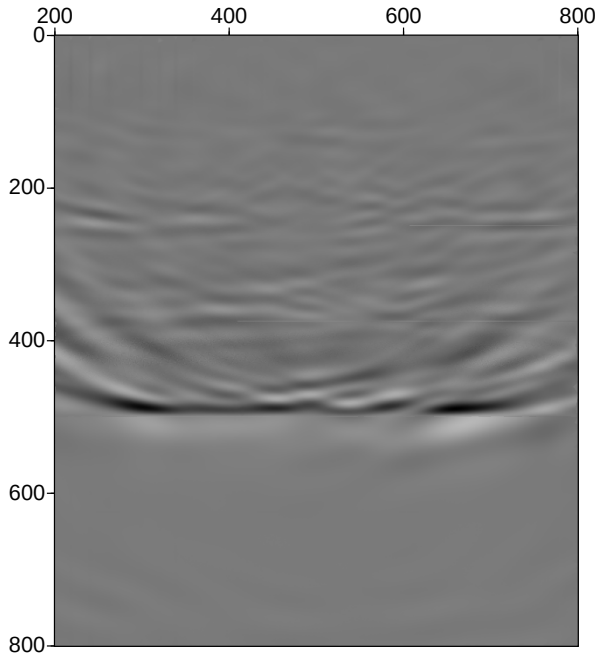


Acoustic

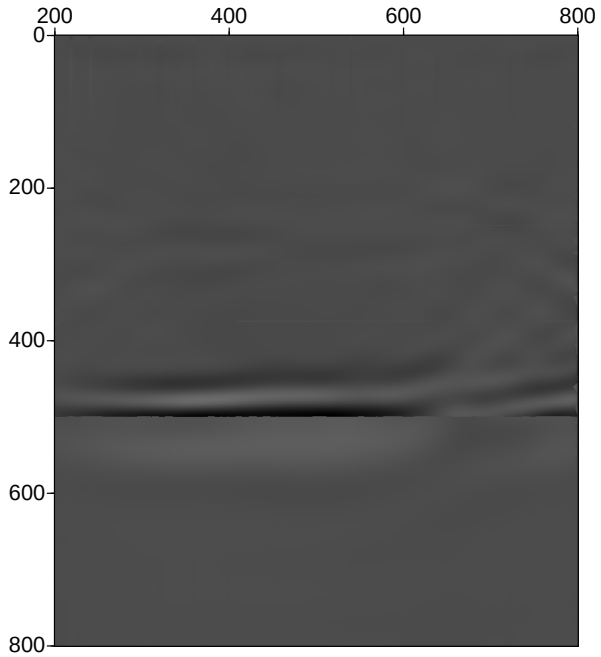


Elastic

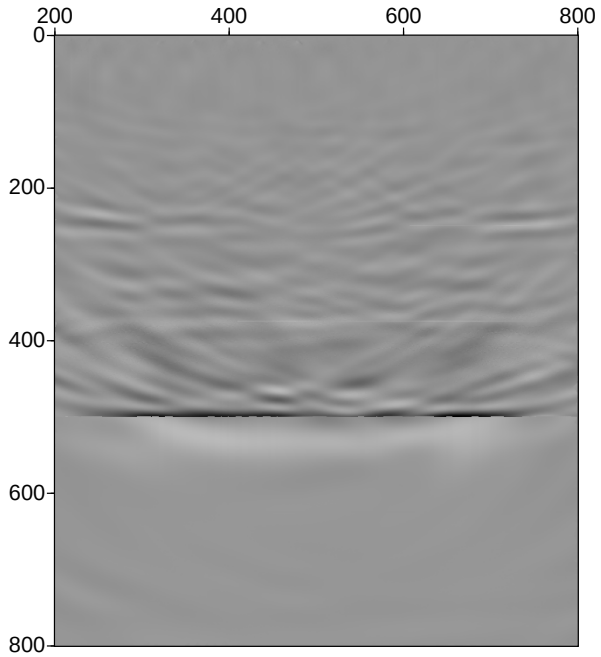
Kernel K_ρ



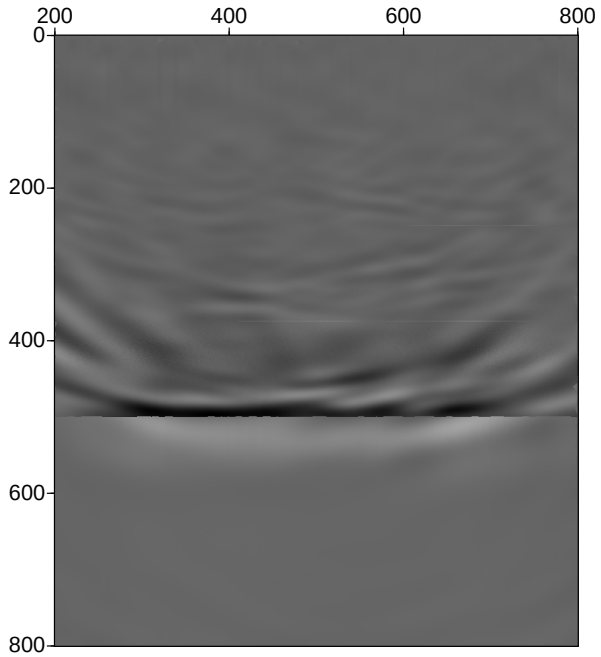
Kernel K_κ



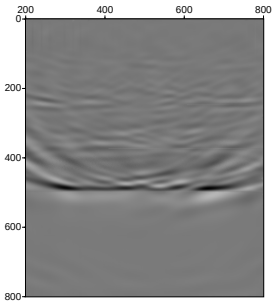
Kernel K_μ



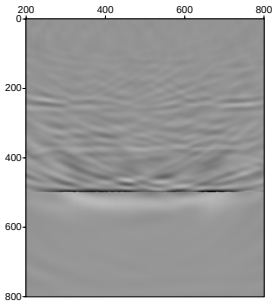
Kernel K



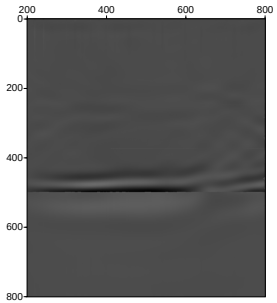
$K_\rho, K_\mu, K_\kappa, K$



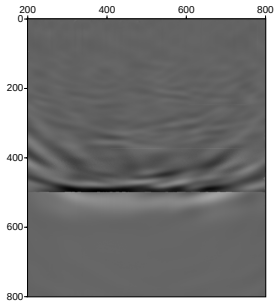
K_ρ



K_μ



K_κ



K

Concluding Remarks

- We have reduced the occupation of the memory without generating additional computational costs
- We have proposed a new imaging condition

Ongoing work

- Application to realistic cases
- Analysis of the impact of elastic condition on the images

Thank you for your attention
Gràcies per la seva atenció
Gracias por su atención